How to get your Devicetree schema bindings accepted in less than 10 iterations

Krzysztof Kozlowski
Qualcomm Landing Team, Linaro
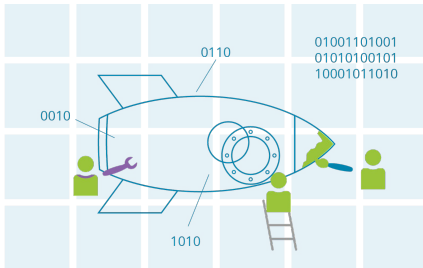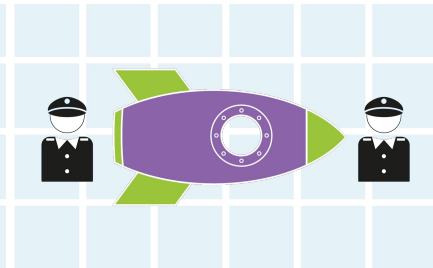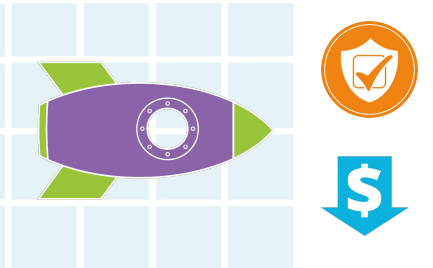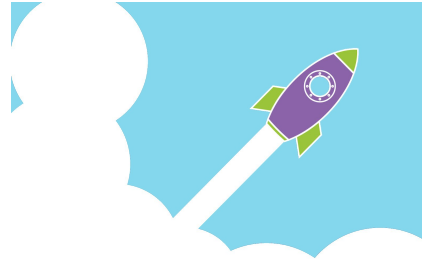krzysztof.kozlowski@linaro.org

September, 2022

Introduction

- Krzysztof Kozlowski
- I am the co-maintainer (with Rob) of Devicetree bindings in Linux kernel
- I also maintain other Linux kernel pieces
  - Memory controller drivers
  - NFC subsystem
  - Samsung Exynos SoC architecture
- I work for Linaro
  - Qualcomm Landing Team / Linaro Developer Services
  - Upstreaming Qualcomm ARM/ARM64 SoCs

Linaro Developer Services

Linaro Developer Services helps companies build, deploy and maintain products on Arm

Arm Software expertise	Specialists in TEE on Arm	Continuous Integration through LAVA	Build, Test and deploy faster
As part of Linaro, Developer Services has some of the world's **leading Arm Software experts**.	We specialize in security and Trusted Execution Environment (TEE) on Arm.	We offer continuous integration (CI) and automated validation through LAVA (Linaro's Automation & Validation Architecture)	We support every aspect of product delivery, from building secure board support packages (BSPs), product validation and long-term maintenance.

For more information go to: https://www.linaro.org/services/

Linaro
Developer Services

Agenda

1. Introduction
2. Bindings and DT schema - short intro
3. Generic rules for bindings
4. DT schema - usage
5. Dos and Don'ts
6. Reusable patterns (reference)
7. References
8. Q&A

Linaro
Developer Services

Disclaimer

- Devicetree and its bindings are not necessarily tied to specific system, however the talk focuses on Linux kernel Devicetree bindings and its DT schema format
- The guidelines here are based on my experience - received and given reviews
- Due to time constraints material presented here is simplified, thus when it contradicts Linux kernel documentation or DT bindings maintainer decision, the latter takes precedence

Bindings and Devicetree schema

- While Devicetree sources (DTS) describe the hardware, the bindings describe the rules how DTS should be constructed
- Therefore the bindings also reference the hardware, not the chosen software implementation (e.g. Linux drivers)
  - Documenting the interface for different implementations
- Previously, bindings in Linux kernel were written in text, without any specific format
- Devicetree schema (DT schema) is the new format which allows:
  - Validation of the bindings itself against meta-schema
  - Validation of the DTS against bindings
- All new bindings must come in DT schema
- Changes to existing TXT bindings:
  - Adding compatibles allowed
  - Adding properties not allowed, please convert the bindings to DT schema first

Example DTS and DT schema

```
spi {
        #address-cells = <1>;
        #size-cells = <0>;

        adc@0 {
                compatible = "adi,ad7291";
                reg = <0>;
                vref-supply = <&adc_vref>;
        };
};
```

```
(...)
title: AD7291 8-Channel, I2C, 12-Bit SAR ADC with Temperature Sensor
description: |
  Analog Devices AD7291 8-Channel I2C 12-Bit SAR ADC with Temperature
Sensor

properties:
  compatible:
    const: adi,ad7291

  reg:
    maxItems: 1

  vref-supply:
    description: The regulator supply for ADC reference voltage.

required:
  - compatible
  - reg
additionalProperties: false
```

Linaro
Developer Services

Generic rules for bindings

- Most of the rules are already covered by
  https://www.kernel.org/doc/html/latest/devicetree/bindings/writing-bindings.html
- Just like DTS, the Devicetree bindings describe the hardware, not the software implementation (e.g. Linux kernel drivers)
  - Bindings are independent of implementation
  - Might be used in several other projects
  - Avoid using Linuxisms (Linux-specific subsystem naming)
- Dual license (GPL-2.0-only OR BSD-2-Clause)
- Bindings filename based on the compatible
  - vendor,device.yaml
  - vendor,soc-ip.yaml
- Bindings headers are also part of the bindings
  - Dual license
  - vendor,device.h
- Bindings changes should not be mixed with a driver code in one patch
  - Sent bindings as separate patches, first in the patchset

Generic rules for bindings - compatibles

- Compatible
  - Should be specific - no wildcards
  - No bus-suffixes
    - Bad: vendor,foo-spi
    - Good: vendor,foo
- syscon and simple-mfd require device-specific compatible as first one
  - `"qcom,sc7280-tcsr", "syscon";`
  - `"qcom,qcs404-imem", "syscon", "simple-mfd";`
- simple-mfd
  - "simple-mfd" means "there's nothing in this node that any of the child nodes depend on"
  - Usually means device is simple, e.g. does not have any properties except children

Generic rules for bindings

- Focus on hardware characteristics and features, instead of specific device programming model
  - Good:
    - regulator-min-microvolt
      (which uses uV as units)
  - Bad:
    - Using device register values to encode logical unit:

```
regulator-min-voltage:
  $ref: /schemas/types.yaml#/definitions/uint32
  enum:
    - 1   # 0.2 V
    - 2   # 0.3 V
    - 3   # 0.4 V
    - 4   # 0.5 V
```

Linaro
Developer Services

DT schema - usage

- There is a guide with an example:
  - https://www.kernel.org/doc/html/latest/devicetree/bindings/writing-schema.html
- DT schema is written in YAML (using json-schema vocabulary)
- Typical usage:

```
pip3 install dtschema

# test the bindings:
make dt_binding_check DT_SCHEMA_FILES=trivial-devices.yaml

# check all the DTSes against given bindings:
export ARCH=arm64 … # cross compile for your arch
make dtbs_check DT_SCHEMA_FILES=trivial-devices.yaml

# check one DTS against given bindings:
make DT_SCHEMA_FILES=trivial-devices.yaml CHECK_DTBS=y qcom/sm8450-hdk.dtb
```

Linaro
Developer Services

Dos and Don'ts

Linaro
Developer Services

Dos and Don'ts - properties

- If there is standard property - use it
  - Look for existing ones in:
    - https://github.com/devicetree-org/dt-schema/tree/main/dtschema/schemas
    - Documentation/devicetree/bindings/gpio/gpio-consumer-common.yaml
    - Other bindings, especially common parts
- Custom properties require:
  - Vendor prefix (foo,property-name)
  - Type ($ref), unless standard unit (see later)
  - Description - describe the feature or hardware, not the Linux driver behavior

```
qcom,avg-samples:
    $ref: /schemas/types.yaml#/definitions/uint32
    description:
        Number of samples to be used for measurement.
```

Dos and Don'ts - no need for types

- No need for type ($ref) for properties:
  - With standard unit suffixes do not need a type ($ref)
    dtschema/schemas/property-units.yaml
  - Described by core schema

```
entry-latency-us:
  description:
    Worst case latency in microseconds required to enter
    the idle state.
```

```
a2vdd-supply:
  description: A 1.8V supply that powers up the A2VDD pin.
```

```
interrupts:
  maxItems: 1
```

Linaro
Developer Services

Dos and Don'ts - arrays

- Items of reg, clocks, dmas, interrupts, resets and others are always strictly ordered
  - The xxx-names (e.g. clock-names) are only helpers
  - Don't use clk/irq suffix in names: "tx" instead of "txirq"
  - Both of these properties (xxx and xxx-names) must have strict constraints on size and order of items
- Declare the items via a list with descriptions

```
clocks:
  items:
    - description: 24 MHz reference
    - description: bus clock
clock-names:
  items:
    - const: ref
    - const: bus
```

Dos and Don'ts - arrays continued

- If minItems==maxItems, only maxItems is enough (although not necessarily in allOf:if:then block)

```
resets:
  maxItems: 2
  description: phandles to the reset lines for both SATA bridges
reset-names:
  items:
    - const: sata0
    - const: sata1
```

- Use maxItems:X for obvious cases (or if xxx-names describes the items)

```
reg:
  maxItems: 1
```

Linaro
Developer Services

Dos and Don'ts - syscon phandles

- Phandle to syscon device requires a vendor, descriptive name and a description
  - Bad:

    ```
    syscon:
      $ref: /schemas/types.yaml#/definitions/phandle
    ```

  - Good:

    ```
    samsung,sysreg:
      $ref: /schemas/types.yaml#/definitions/phandle
      description: Phandle to System Register syscon
    ```

Dos and Don'ts - syscon phandles continued

- ● Best example:

```
samsung,sysreg:
  $ref: /schemas/types.yaml#/definitions/phandle-array
  items:
    - items:
        - description: phandle to System Register syscon node
        - description: offset of SW_CONF register for this controller
  description:
    The phandle to System Register syscon node for the same
    domain where this USI controller resides
```

Linaro
Developer Services

Dos and Don'ts - additional/unevaluated

- additionalProperties and unevaluatedProperties control how other properties are treated (the ones not mentioned in the current schema)
- Most cases: choose either additionalProperties or unevaluatedProperties and set it to *false*
- If schema does not reference any other schema:

```
properties:
  ...


required:
  ...


additionalProperties: false
```

Linaro
Developer Services

Dos and Don'ts - additional/unevaluated

- If schema references other schema, you can list applicable properties from other schema and do not allow anything else:

```
allOf:
  - $ref: panel-common.yaml#

properties:
  backlight: true          # coming from panel-common.yaml
  reset-gpios: true         # coming from panel-common.yaml
  ...

required:
  ...

additionalProperties: false
```

Linaro
Developer Services

Dos and Don'ts - additional/unevaluated

- Allow all fields from the other schema
  - This is preferred if the referenced schema is in general valid for your device and any of its properties can be applicable, e.g. regulator.yaml

```
patternProperties:
  "^LDO[1-3]$":
    type: object
    $ref: regulator.yaml#
    unevaluatedProperties: false
```

```
allOf:
  - $ref: panel-common.yaml#
properties:
...
unevaluatedProperties: false
```

Linaro
Developer Services

Dos and Don'ts - examples

- Example is used to validate the DT schema
- Include useful DTS example(s)
  - …but not 10 examples with different only compatibles
- Use 2- or 4-space indentation for DTS example
  - 4-space is preferred, nicely aligns with the opening -|

```
examples:
  - |
    adc@0 {
        compatible = "adi,ad7190";
        reg = <0>;
    };
```

Linaro
Developer Services

Dos and Don'ts - examples continued

- No "status=okay/disabled" in the examples
- No unnecessary consumer examples inside provider bindings (e.g. clock controllers)
  - In that context, the usage of consumer is obvious
  - Not related to particular provider
- Device node names should be generic ("adc", not "ad7190")
  - Devicetree spec: 2.2.2. Generic Names Recommendation

Reusable patterns (reference)

Linaro
Developer Services

Reusable patterns (reference)

Property required and present only in one variant:

```
allOf:
  - if:
      properties:
        compatible:
          contains:
            const: vendor,soc2-ip
    then:
      required:
        - foo-supply
    else:       # If otherwise the property is not allowed:
      properties:
        foo-supply: false
```

Linaro
Developer Services

Reusable patterns (reference)

Excluding properties, but one is required:

```
oneOf:
  - required:
      - reg

  - required:
      - size
```

Linaro
Developer Services

Reusable patterns (reference)

Excluding properties and none is required:

```
allOf:
  - if:
      required:
        - s5m8767,pmic-buck2-uses-gpio-dvs
    then:
      properties:
        s5m8767,pmic-buck3-uses-gpio-dvs: false
  - if:
      required:
        - s5m8767,pmic-buck3-uses-gpio-dvs
    then:
      properties:
        s5m8767,pmic-buck2-uses-gpio-dvs: false
```

Linaro
Developer Services

Reusable patterns (reference)

Array of integers with some constraints (e.g. min/max values):

```
properties:
  vendor,int-array-variable-length-and-constrained-values:
    description: Array might define what type of elements might be used
    $ref: /schemas/types.yaml#/definitions/uint32-array

    minItems: 2

    maxItems: 3

    items:

      minimum: 0

      maximum: 8
```

Linaro
Developer Services

Reusable patterns (reference)

Variable length arrays (per variant):

```
properties:
  clocks:
    minItems: 2
    maxItems: 4
  clock-names:
    minItems: 2
    maxItems: 4

...
```

```
allOf:
  - if:
      properties:
        compatible:
          contains:
            const: foo,bar
    then:
      properties:
        clocks:
          minItems: 4
        clock-names:
          items:
            - description: ...
```

Linaro
Developer Services

Reusable patterns (reference)

- [Dependency between properties](#)
- [Restricting property based on other one](#)
- [Variable length of array - last interrupt optional](#)
- [Phandle to syscon with offset](#)
- [uint32 matrix, variable length of two-items tupples](#)
- [$ref depending on compatible](#)

References

- Writing bindings:
  https://www.kernel.org/doc/html/latest/devicetree/bindings/writing-bindings.html
- Writing DT schema:
  https://www.kernel.org/doc/html/latest/devicetree/bindings/writing-schema.html
- Example schema:
  https://www.kernel.org/doc/html/latest/devicetree/bindings/writing-schema.html#example-schema
- Standard property types/suffixes:
  https://github.com/devicetree-org/dt-schema/blob/main/dtschema/schemas/property-units.yaml
- dt-schema core schemas:
  https://github.com/devicetree-org/dt-schema/tree/main/dtschema/schemas

Introducing Linaro

Linaro collaborates with businesses and open source communities to:

- Consolidate the Arm code base & develop common, low-level functionality

- Create open source reference implementations & standards

- Upstream products and platforms on Arm

Why do we do this?

- To make it easier for businesses to build and deploy high quality and secure Arm-based products

- To make it easier for engineers to develop on Arm

Two ways to collaborate with Linaro:

1. Join as a member and work with Linaro and collaborate with other industry leaders

2. Work with Linaro Developer Services on a one-to-one basis on a project

For more information go to: www.linaro.org

Linaro
Developer Services

Linaro membership collaboration

Thank you

Questions?

Linaro
Developer Services